International Journal of Engineering Issues
Vol. 2015, no. 2, pp. 73-78
ISSN: 2458-651X

Infinity Sciences

Modeling Cyber – Physical Systems for Engineering Complex Software

Prof. T V Gopal
Department of Computer Science and Engineering
Anna University
Chennai – 600 025, India
Email: *gopal@annauniv.edu*

Abstract- Engineering Disciplines have principles based on the laws of physics, biology, chemistry, or mathematics. Principles become practice through methods and techniques. The methods and techniques are often packaged in a methodology that can be enforced by tools. Software engineering is different. It had to evolve its principles based solely on observations of thousands of projects. Every claim needs to be validated separately. In the absence of generic principles grounding a given claim within a given context becomes a real challenge. Computer Science has an inherent credibility problem due the imperative need for blending both the natural and the artificial. This paper is on modeling the Cyber - Physical Systems. It is based on mathematical foundations for the evolving dynamics of design, development and use of complex software. Geometric Algebra and Grammar of Graphics are chosen to provide better visualization of the Cyber – Physical Systems.

Keywords**:** *Cyber – Physical Systems, Complexity, Physics, Computing, Computational Models, Geometric Algebra, Grammar of Graphics*

I. INTRODUCTION

"We all want our software systems to be fast, reliable, easy to use, readable, modular, structured and so on. But these adjectives describe two different sort of qualities. On one side we are considering such qualities as speed or ease of use, whose presence or absence in a software product may be detected by its users. These properties may be called external quality factors. Other qualities applicable to a software product, such as being modular, or readable, are internal factors, perceptible only to computer professionals who have access to the actual software text. In the end, only external factors matter. If I use a Web browser or live near a computer-controlled nuclear plant, little do I care whether the source program is readable or modular if graphics take ages to load, or if a wrong input blows up the plant. But the key to achieving these external factors is in the internal ones: for the users to enjoy the visible qualities, the designers and implementers must have applied internal techniques that will ensure hidden qualities." **- Bertrand Meyer, 1997**

Edsger Wybe Dijkstra defined "intellectual distance" as the distance between the real world problem and the computerized solution to the problem. Programmers have always known about civil, electrical, and computer engineering. However, what engineering might mean for software remains a debate till date. The "intellectual distance"

between the design and use of the complex software in a given context needs to be minimized in a well-engineered and verifiable manner.

Software Engineering depends on strong abstractions. Abstraction enables more focus on the concepts apart from particular instances of the concepts. Abstraction facilitates the postponement of some structural considerations and many algorithmic details for a later expansion. It reduces the amount of complexity [8] that must be dealt with at any given point. Design is a process of proceeding from abstraction to concrete representations. Various models help to communicate different aspects of the system by masking some of the complexity and revealing some relationships. In the absence of generic principles the construction of evolving complex software is based on a series of claims that are very difficult to verify and validate.

Any given principle is a proposal formulated in a prescriptive way. It should not be directly associated with, or arise from, a particular technology, a very specific method, or a Unique technique. It should not be an activity of software engineering. The principle should not dictate a compromise (or a proportioning) between two actions or concepts. A principle of software engineering should include concepts connected to the engineering discipline. It must be possible to test the formulation of a principle in practice, or to check its consequences.

The context for constructing the complex software is a Cyber – Physical System [5]. Cyber-Physical Systems (CPS) are integrations of computation, networking, and physical processes. CPS facilitates the necessary abstractions for modeling and designing the integrated whole. Cyber-physical systems (CPS) are physical and well-engineered systems that are monitored, coordinated, controlled and integrated by a computing and communication core with complex software connecting the various sub-systems.

"We all realize that we're kind of surrounded with technology: there's little device here recording us, there's tables, chairs, spoons, light bulbs. Each of these things seem pretty mechanical, pretty inert in a certain sense, not very interactive, you know, a hammer, roads. But each one of these technologies actually requires many other technologies to make and produce. So your little thing in your pocket that you use for a phone might require thousands of other technologies to create it and support it and keep it going, and each of those technologies may require hundreds of thousands of subtechnologies below it. And that network of different technologies and the co-dependency that each of those technologies have on each other forms a virtual organism, a super organism.

We can keep stepping back and realize that all these technologies are in some ways co-dependent and related and connected to each other in some way and that largest of all the networks of all these technologies together I call the Technium." **– Kevin Kelly**

The author defines the technium as a Cyber – Physical Systems. CPS represent a conglomeration of technologies in embedded systems, distributed systems, dependable systems, real-time systems, communication systems with advances in energy-efficient networking, microcontrollers, sensors and actuators.

II. THE DYNAMICS OF CYBER – PHYSICAL SYSTEMS

Classical mechanics studies the mechanical parts that move. The techniques used to study the dynamics of such parts are also applicable to other physical systems, including circuits, chemical processes, and biological processes. It is thus possible to arrive at a set of principles across multiple engineering disciplines to engineer mechanical parts. Mechanical parts are easiest for most people to visualize and are also easily engineered. The motion of mechanical parts can often be represented using differential equations, or equivalently, integral equations. Such representations work well for motion that is amenable for precise definition using the notions of linearity, time invariance, and continuity. However, discrete events are necessary for other motions. Continuous components evolve smoothly, while discrete components evolve abruptly. Finite state machines are used to represent discrete dynamics.

Theory of Computation in its present form begins with the Finite State Machines and culminates in the Turing Machine. Finite state machines describe the class of regular languages, have no memory and has limited number of states. Turing Machines describe a much larger class of languages and have a much larger computational power. There are obvious differences between a Turing machine and a real computer [1]. The computer is finite in size, prone to

failures and it is made from decaying matter. There are many emerging models of computation such as Spintronics that attempt to harness the matter that forms the substrate for the computation. The concept of an automaton is changing to include the theories related to cybernetics along with the present automata theory.

The physical system is given by a differential or an integral equation [10] that relates input signals (force or torque) to output signals (position, orientation, velocity, or rotational velocity). Such a physical system can be viewed as a component in a larger system. Cyber – Physical systems include both continuous and discrete components.

Computation needs to be organized as a network of interconnected components of some kind, each of which is free to run when it pleases, propagating information around the network as proves possible. The consequence that the structure of the CPS [7] does not impose a temporal ordering. Instead each component in the CPS functions as it pleases, and allows the order of operations to be determined by the needs of the solution and not on the structure of the problem description. Constructing well-engineered software for CPS thus needs different foundational principles.

III. VECTORS, GEOMETRY AND ALGEBRA

Visualization of the solution is the key to success in CPS. Drawing is a proven method across engineering disciplines to facilitate accurate and clear communication of the proposed solution. However, present day software engineering has evolved many diagrammatic representations based on the automata theory that are highly restrictive. Also, the notations for the diagrams are usually not interoperable with other engineering disciplines. The control switches between propagation of arbitrary partial information and search for alternative paths of execution only when a contradictions are discovered. The primary concern is “evaluation” to quickly discover any contradiction to ensure faster execution by choosing the most optimal paths of computation. The over-rigid notions of computational timing and constraint make the methods archaic for CPS [4,9].

Time is nature’s way of keeping everything from happening at once. Space is nature’s way of keeping everything from happening in the same place. Physics has limitations imposed by the very nature of physical reality. Mathematics may provide both natural and artificial solutions warranted by the complex software. However, the mathematics applied to physics must be limited, simply by the limits imposed by physics.

The foundational point of using Calculus is for an easy mathematical approach for the most reasonable estimation of Time and Space complexity [6] for a computation. “Algebra” connotes “relationships”. “Linear Algebra” indicates “line-like relationships” that are predictable. There are many algebras. Boolean Algebra that made computation with numbers happen using machines that are physically realizable using the logic using the symbols 0 ad 1.

Vectors are "numbers with direction". For the physical phenomena, such as velocity and displacement, vectors are very useful representations. Geometry and Trigonometry are very difficult to apply in many contexts. Vector algebra was invented in order to solve two-dimensional and three-dimensional problems without the use of cumbersome geometry. The calculus of real numbers, vector calculus and complex analysis eventually paved way for Geometric Calculus. This calculus serves better to understand the intricacies of electrodynamics [11]. Geometric algebra [2] and its extension to geometric calculus unify, simplify, and generalize many areas of mathematics that involve geometric ideas. They also provide a unified mathematical language for physics, engineering, and the geometrical aspects of computer science such as those found in graphics, robotics and computer vision.

The following aspects [3] of Evolving Complex Software Systems thus become more amenable to analysis and design.

- Interconnected and interdependent elements and dimensions
- Feedback processes promote and inhibit change within systems
- System characteristics and behaviors emerge from simple rules of interaction
- Nonlinearity
- Sensitivity to initial conditions

- Phase space – the 'space of the possible'
- Attractors, chaos and the 'edge of chaos'
- Adaptive agents
- Self-organization
- Co-evolution

IV. GEOMETRIC ALGEBRA

A vector is "anything that can be represented by arrows that add head-to-tail." Such objects have magnitude (how long is the arrow) and direction (which way does it point). Real numbers have two analogous properties: a magnitude (absolute value) and a sign (plus or minus). Higher-dimensional objects in real vector spaces also have these properties: for example, a surface element is a plane with a magnitude (area) and an orientation (clockwise or counterclockwise). If we associate real scalars with zero-dimensional spaces, then we can say that scalars, vectors, planes and so on, have three features in common:

- **An attitude:** exactly which subspace is represented?
- **A weight:** an amount, or a length, area, volume and so on.
- **An orientation:** positive or negative, forward or backward, clockwise or counterclockwise. No matter what the dimension of the space, there are always only two orientations.

Geometric objects (points, lines, planes, circles and so on) are represented by members of an algebra, a geometric algebra, rather than by equations relating coordinates. Geometric operations on the objects (rotate, translate, intersect, project, construct the circle through three points, and so on) are then represented by algebraic operations on the objects. Geometric algebra is coordinate-free i.e coordinates are needed only when specific objects or operations are under consideration. The author opines that Geometric Algebra provides a better visualization of the CPS and the pertinent dynamics. Grammar of Graphics [13] is chosen to lend expressions to the Geometric Algebra of the CPS.

V. GRAMMAR OF GRAPHICS

A grammar provides the fundamental principles or rules of an art or science. A good grammar will allow us to gain insight into the composition of complicated graphics, and reveal unexpected connections between seemingly different graphics. A grammar provides a strong foundation for understanding a diverse range of graphics. A grammar may also help guide us on what a well-formed or correct graphic looks like, but there will still be many grammatically correct but nonsensical graphics [6]. The proposed block schematic for visualizing a Cyber – Physical System is given in Figure 1.

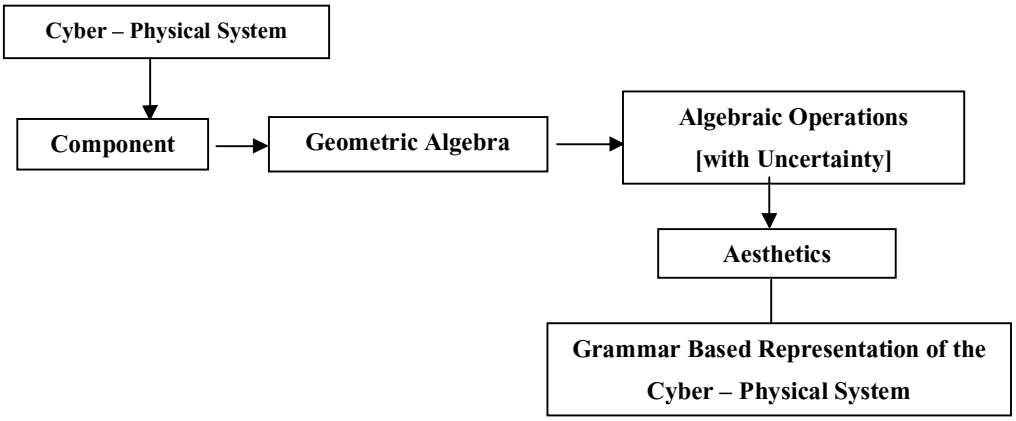

Figure 1. *Proposed Block Schematic for Visualizing a Cyber – Physical System*

The Uncertainty factored in the algebraic operations indicated in Figure 1 above is a combination of Variability, Noise, Incompleteness, Indeterminacy, Bias, Error, Accuracy, Precision, Reliability, Validity, Quality and Integrity. The Aesthetic Attributes pertaining to Figure 1 are indicated in the Table 1.

Table 1. *The Aesthetic Attributes*

Form	Surface	Motion	Sound	Text
Position	Color	Direction	Tone	Label
Size	Texture	Speed	Volume	
Shape	Blur	Acceleration	Rhythm	
Rotation	Transparency		Voice	
Resolution				

Positioning of the Components may be a separate layer [12] based on the Calculus of Geometry [3].

VI. CONCLUSIONS

This paper proposes a mathematical approach to architect the automata that model the Cyber – Physical Systems and hence paving way for better visualization and representation for the 14 Grand Challenges in Engineering [14] mentioned below.

1. Make solar energy economical
2. Provide energy from fusion
3. Develop carbon sequestration methods
4. Manage the nitrogen cycle
5. Provide access to clean water
6. Restore and improve urban infrastructure
7. Advance health informatics and practice of Medicine
8. Engineer better medicines
9. Reverse-engineer the brain
10. Prevent nuclear terror
11. Secure cyberspace with methods better than the traditional practices
12. Enhance virtual reality
13. Advance personalized learning
14. Engineer the tools of scientific discovery

ACKNOWLEDGEMENTS

The author places on record his sincere thanks to Anna University, Chennai, the members of the Steering Committee of the Series of Annual Conferences on Theory and Applications of Models of Computation [TAMC] and the Conference Chairs of the IEEE Series of Conferences on Norbert Wiener in the 21st Century.

REFERENCES

[1] DAVID, Evans. Introduction to Computing Explorations in Language, Logic, and Machines. CreateSpace Independent Publishing Platform, 2011.

[2] DORAN, Chris et LASENBY, A. N. Geometric algebra for physicists. Cambridge : Cambridge University Press, 2003.

[3] GOPAL T,V. The Physics of Evolving Complex Software Systems. International Journal of Engineering Issues, Vol. 2015, no. 1, p. 33-39.

[4] GLOTFELTER, Paul, EICHELBERGER, Travis, et MARTIN, Patrick J. PhysiCloud: A cloud-computing framework for programming cyber-physical systems. Report supported by US National Science Foundation through grant CNS-1239221, 2015.

[5] IVAN, Ruchkin, BRADLEY, Schmerl, et DAVID, Garlan. Analytic Dependency Loops in Architectural Models of Cyber-Physical Systems. In: *8th International Workshop on Model-based Architecting of Cyber-Physical and Embedded Systems (ACES-MB)*. Ottawa, Canada, 2015.

[6] IVOR, Grattan-Guinness. The Scope and Limitations of Algebras: Some Historical and Philosophical Considerations. Theology and Science, 2011, Vol. 9, no. 1, p 137 – 147.

[7] LEE, Edward Ashford et SESHIA, Sanjit A. Introduction to embedded systems. Morrisville, NC : LeeSeshia.org., 2011.

[8] MAINZER, Klaus. Thinking in complexity. Berlin : Springer-Verlag, 2014.

[9] RAGUNATHAN (Raj) Rajkumar et al. Cyber-Physical Systems: The Next Computing Revolution. In: Design Automation Conference 2010, Anaheim, California, USA.

[10] RAJARAMAN, V. Analog computation and simulation. New Delhi : Prentice-Hall of India, 1974.

[11] TERJE, G, Vold. An Introduction to Geometric Calculus and its Application to Electrodynamics. *American Journal of Physics*, 2013, Vol. 61, no, 6, p 505 – 513..

[12] WICKHAM Hadley. A Layered Grammar of Graphics. Report Supported by the National Science Foundation under grant 0706949, 2008.

[13] WILKINSON, Leland et WILLS, Graham, 2005, The grammar of graphics. New York : Springer.

[14] WILLIAM Perry et al. Grand Challenges in Engineering. Report by National Academy of Engineering, 2008.